TRP++: A temporal resolution prover*

Ullrich Hustadt and Boris Konev**

Logic and Computation Group, Department of Computer Science
University of Liverpool, Liverpool L69 7ZF, UK
{U.Hustadt, B.Konev}@csc.liv.ac.uk

1 Introduction

Temporal logics are extensions of classical logic, with operators that deal with time. They have been used in a wide variety of areas within Computer Science and Artificial Intelligence, for example robotics [27], databases [29], hardware verification [15] and agent-based systems [24]. In particular, propositional temporal logics have been applied to:

– the specification and verification of reactive (e.g. distributed or concurrent) systems [22];
– the synthesis of programs from temporal specifications [21, 23];
– the semantics of executable temporal logic [10, 11];
– algorithmic verification via model-checking [4, 14]; and
– knowledge representation and reasoning [1, 8, 30].

In developing these techniques, temporal proof is often required, and we base our work on practical proof techniques for the clausal resolution method for propositional linear-time temporal logic PLTL. The method is based on an intuitive clausal form, called SNF, comprising three main clause types and a small number of resolution rules [12]. While the approach has been shown to be competitive [16, 17] using a prototype implementation of the method, we now aim at an even more efficient implementation. This implementation, called **TRP++**, is the focus of this paper.

2 Basics of PLTL

Let P be a set of propositional variables. The set of formulae of *propositional linear time logic* PLTL (over P) is inductively defined as follows: (i) $\top$ is a formula of PLTL, (ii) every propositional variable of P is a formula of PLTL, (iii) if φ and ψ are formulae of PLTL, then $\neg\varphi$ and $(\varphi \lor \psi)$ are formulae of PLTL, and (iv) if φ and ψ are formulae of PLTL, then $\bigcirc\varphi$ (in the next moment of time φ is true), $\Diamond\varphi$ (sometimes in the future φ is true), $\Box\varphi$ (always in the future φ is true), $(\varphi\,\mathcal{U}\,\psi)$ (φ is true until ψ is true), and $(\varphi\,\mathcal{W}\,\psi)$ (φ is true unless ψ is true) are

* Work supported by EPSRC grant GR/L87491.
** On leave from Steklov Institute of Mathematics at St.Petersburg

formulae of PLTL. Other Boolean connectives including $\bot$, $\wedge$, $\rightarrow$, and $\leftrightarrow$ are defined using $\top$, $\neg$, and $\vee$.

PLTL-formulae are interpreted over ordered pairs $\mathcal{I} = \langle \mathcal{S}, \iota \rangle$ where (i) $\mathcal{S}$ is an infinite sequence of *states* $(s_i)_{i \in \mathbb{N}}$ and (ii) ι is an *interpretation function* assigning to each state a subset of P.

We define a binary relation $\models$ between a PLTL-formula φ and a pair consisting of a PLTL-interpretation $\mathcal{I} = \langle \mathcal{S}, \iota \rangle$ and a state $s_i \in \mathcal{S}$ as follows.

$$
\begin{array}{lll}
\mathcal{I}, s_i \models p & \text{iff } p \in \iota(s_i) & \mathcal{I}, s_i \models \top \\
\mathcal{I}, s_i \models (\varphi \vee \psi) & \text{iff } \mathcal{I}, s_i \models \varphi \text{ or } \mathcal{I}, s_i \models \psi & \mathcal{I}, s_i \models \neg\varphi \text{ iff } \mathcal{I}, s_i \not\models \varphi \\
\mathcal{I}, s_i \models \bigcirc \varphi & \text{iff } \mathcal{I}, s_{i+1} \models \varphi & \\
\mathcal{I}, s_i \models \Box \varphi & \text{iff for all } j \in \mathbb{N},\ j \geq i \text{ implies } \mathcal{I}, s_j \models \varphi & \\
\mathcal{I}, s_i \models \Diamond \varphi & \text{iff there exists } j \in \mathbb{N} \text{ such that } j \geq i \text{ and } \mathcal{I}, s_j \models \varphi & \\
\mathcal{I}, s_i \models (\varphi \,\mathcal{U}\, \psi) & \text{iff there exists } j \in \mathbb{N} \text{ such that } j \geq i,\ \mathcal{I}, s_j \models \psi, \text{ and} & \\
 & \quad \text{for all } k \in \mathbb{N},\ j > k \geq i \text{ implies } \mathcal{I}, s_k \models \varphi & \\
\mathcal{I}, s_i \models (\varphi \,\mathcal{W}\, \psi) & \text{iff } \mathcal{I}, s_i \models \varphi \,\mathcal{U}\, \psi \text{ or } \mathcal{I}, s_i \models \Box \varphi &
\end{array}
$$

If $\mathcal{I}, s_i \models \varphi$ then we say φ is *true*, or *holds*, at s_i in $\mathcal{I}$. An interpretation $\mathcal{I}$ *satisfies* a formula φ iff φ holds at s_0 in $\mathcal{I}$ and it *satisfies* a set N of formulae iff for every formula $\psi \in N$, $\mathcal{I}$ satisfies ψ. In this case, $\mathcal{I}$ is a *model* for φ and N, respectively, and we say φ and N are (PLTL-)satisfiable. The satisfiability problem of PLTL is known to be PSPACE-complete [28].

Arbitrary PLTL-formulae can be transformed into *separated normal form* (SNF) in a satisfiability equivalence preserving way using a renaming technique replacing non-atomic subformulae with new propositions and removing all occurrences of the $\mathcal{U}$ and $\mathcal{W}$ operator [9, 12].

The result is a set of *SNF clauses* of the following form (which differs slightly from [9, 12]).

$$
\begin{array}{ll}
\bigvee_{i=1}^{n} L_i & \text{(initial clause)} \\
\Box(\bigvee_{j=1}^{m} K_j \vee \bigvee_{i=1}^{n} \bigcirc L_i) & \text{(global clause)} \\
\Box(\bigvee_{j=1}^{m} K_j \vee \Diamond L) & \text{(eventuality clause)}
\end{array}
$$

Here, K_j, L_i, and L (with $1 \leq j \leq m$, $0 \leq m$, and $1 \leq i \leq n$, $0 \leq n$) denote propositional literals.

In the following, we assume that SNF clauses are sets of (temporal) literals. Furthermore, if $C = L_1 \vee \ldots \vee L_n$ we use $\bigcirc C$ to denote $\bigcirc L_1 \vee \ldots \vee \bigcirc L_n$.

3 Clausal temporal resolution

TRP++ is based on the resolution method for PLTL proposed by Fisher [9] (see also [6, 7, 12]) which involves the translation of PLTL-formulae to separated normal form, classical resolution within states (known as initial and step resolution) and temporal resolution over states between eventuality clauses containing a literal like $\Diamond \neg p$ and global clauses that together imply $\Box p$ (known as eventuality resolution). Figure 1 contains a list of all the inference rules. To simplify the presentation, we have represented the conclusion of the eventuality resolution rule as a PLTL-formula (which would have to be transformed into a set of SNF clauses). In our implementation, we directly produce the corresponding SNF clauses without reverting to the transformation procedure. It should be obvious that finding a set of SNF clauses which satisfies the side conditions of the eventuality resolution rule is a non-trivial problem.

These inference rules provide a sound and complete calculus for deciding the satisfiability of a set N of SNF clauses. Furthermore, under the assumption that an inference step is performed only once for the same set of premises (or that we stop as soon as no new SNF clauses can be derived), any derivation from a set N of SNF clauses will always terminate. Since any PLTL-formula can be transformed into a satisfiability equivalent set of SNF clauses, this means that the combination of this transformation process and the temporal resolution calculus provides a decision procedure for PLTL.

Initial resolution rules:

$$\frac{C_1 \lor L \qquad \neg L \lor C_2}{C_1 \lor C_2} \qquad\qquad \frac{C_1 \lor L \qquad \Box(\neg L \lor D_2)}{C_1 \lor D_2}$$

where $C_1 \lor L$ and $\neg L \lor C_2$ are initial clauses, $\Box(\neg L \lor C_2)$ is a global clause and $\neg L \lor D_2$ is a propositional clause.

Step resolution rules:

$$\frac{\Box(C_1 \lor L) \qquad \Box(\neg L \lor C_2)}{\Box(C_1 \lor C_2)} \qquad \frac{\Box(C_1 \lor L) \qquad \Box(\bigcirc \neg L \lor D_2)}{\Box(\bigcirc C_1 \lor D_2)}$$

$$\frac{\Box(D_1 \lor \bigcirc L) \qquad \Box(\bigcirc \neg L \lor D_2)}{\Box(D_1 \lor D_2)}$$

where $\Box(C_1 \lor L)$ and $\Box(\neg L \lor C_2)$ are global clauses and $C_1 \lor L$ and $\neg L \lor C_2$ are propositional clauses, and $\Box(\bigcirc \neg L \lor D_2)$ and $\Box(D_1 \lor \bigcirc L)$ are global clauses. (The side conditions ensure that no clauses with nested occurrences of the $\bigcirc$-operator can be derived.)

Eventuality resolution rule:

$$\frac{\begin{array}{lll} \Box(C_1^1 \lor \bigvee_{l=1}^{k_1^1} \bigcirc D_{1,l}^1) & \Box(C_1^n \lor \bigvee_{l=1}^{k_1^n} \bigcirc D_{1,l}^n) & \\ \vdots & \vdots & \\ \Box(C_{m_1}^1 \lor \bigvee_{l=1}^{k_{m_1}^1} \bigcirc D_{m_1,l}^1) \ \cdots & \Box(C_{m_n}^n \lor \bigvee_{l=1}^{k_{m_n}^n} \bigcirc D_{m_n,l}^n) & \Box(C \lor \Diamond L) \end{array}}{\Box(C \lor (\neg(\bigvee_{i=1}^n \bigwedge_{j=1}^{m_i} C_j^i) \mathcal{W} L))}$$

where for all i, $1 \leq i \leq n$, $(\bigwedge_{j=1}^{m_i} \bigvee_{l=1}^{k_j^i} D_{j,l}^i) \rightarrow \neg L$ and $(\bigwedge_{j=1}^{m_i} \bigvee_{l=1}^{k_j^i} D_{j,l}^i) \rightarrow (\bigvee_{i=1}^n \bigwedge_{j=1}^{m_i} C_j^i)$ are provable.

Fig. 1. The temporal resolution calculus

4 Implementation details

Figure 2 shows the main procedure of our implementation of the temporal resolution calculus of Section 3 which consists of a loop where at each iteration (i) the set of SNF clauses is saturated under application of the initial and step resolution rules using function *Saturate* shown in Figure 3, and (ii) then for every eventuality clause in the SNF clause set, an attempt is made to find a set of premises for an application of the eventuality resolution rule using function *BFS* shown in Figure 4 which implements Dixon's search algorithm [7]. If we find such a set, the set of SNF clauses representing the conclusion of the application is added to the current set of SNF clauses. The main loop terminates if the empty clause is derived, indicating that the initial set of SNF clauses and the PLTL-formula it is stemming from are unsatisfiable, or if no new clauses have been derived during the last iteration of the main loop, which in the absence of the empty clause indicates that the initial set of SNF clauses and the PLTL-formula it is stemming from are satisfiable. Since the number of SNF clauses which can be formed over the finite set of propositional variables contained in the initial set of SNF clauses is itself finite, we can guarantee termination of the main procedure.

It is easy to check that under the natural arithmetic translation of initial and global clauses into first-order logic (an initial clause $(\neg)q_1 \vee \ldots \vee (\neg)q_n$ is represented by the first-order clause $(\neg)q_1(0) \vee \ldots \vee (\neg)q_n(0)$ where 0 is a constant representing the natural number 0; a global clause $(\neg)p_1 \vee \ldots \vee (\neg)p_m \vee \bigcirc(\neg)q_1 \vee \ldots \vee \bigcirc(\neg)q_n$ as $\forall x\,((\neg)p_1(x) \vee \ldots \vee (\neg)p_m(x) \vee (\neg)q_1(s(x)) \vee \ldots \vee (\neg)q_n(s(x)))$ where s is representing the successor function on the natural numbers), initial and step resolution exactly correspond to usual first-order *ordered* resolution with respect to an atom ordering $\prec$ where $p(x) \prec q(s(x)) \prec r(s(s(x)))$ for arbitrary predicate symbols p, q, and r. Search for premises for the eventuality resolution rule (the computationally most costly part of the method), as implemented in *BFS*, is again based on step resolution. Hence, performance of the step resolution inference engine is critical for the system. Using the arithmetic translation, any state-of-the-art first-order resolution system could perform step resolution (and initial resolution). However, our formulae have a very restrictive nature, and **TRP++** uses its own "near propositional" approach to deal with them.

Data representation. We represent SNF clauses as propositional clauses and supply each literal with an "attribute"—one of initial, global_now, and global_next with obvious meaning (eventuality clauses are kept and processed separately). In addition, we define a total ordering $<$ on attributed literals which satisfies the constraint that for every initial literal K, global_now literal L, and global_next literal M we have $K < L < N$.

The ordering is then used to restrict resolution inference steps to the maximal literals in a clause. This ensures, for example, that in a clause $C \vee L$ where L is a global_now literal but C contains some global_next literals, a resolution step

```
procedure main(N)
begin
  New := N;
  while (⊥ ∉ N and New ≠ ∅) do
    N := Saturate(N);
    if (⊥ ∉ N) then
      New := ∅;
      N_0 := select_global(N);
      foreach □(⋁_{j=1}^m K_j ∨ ◇L) ∈ N do
        G := BFS(N_0, 0, true, L)
        if (G ≠ ∅) then
          New := New ∪ e-res (□(⋁_{j=1}^m K_j ∨ ◇L), G);
        endif
      end
      New := simp(New, N);
      N := N ∪ New;
    endif
  end
end
```

where *select_global*(N) is the set of all global clauses selected from N, *e-res* $(\Box(\bigvee_{j=1}^{m} K_j \vee \Diamond L), G)$ is the set of conclusions of the eventuality resolution rule applied to the eventuality clause $\Box(\bigvee_{j=1}^{m} K_j \vee \Diamond L)$ and the global clauses in G, and *simp*(New, N) is the result of simplification (e.g. by subsumption) of clauses from *New* by clauses from N.

Fig. 2. Main procedure of **TRP++**

on L is impossible. (Note that this behaviour is in accordance with the inference rules of the temporal resolution calculus.)

To simulate the effect of first-order unification on the arithmetical translation of SNF clauses, unification of literals in our “near propositional” representation has to take their attributes into account. For example, it is impossible to unify an initial literal with a global_next literal (since it is impossible to unify a literal $(\neg)p(0)$ with $(\neg)p(s(x))$.) However, it is possible to unify a global_now literal with a global_next literal and the unifying substitution will turn the global_now literal into a global_next literal (since it is possible to unify a literal $(\neg)p(y)$ with $(\neg)p(s(x)))$.) In this case we will also have to turn all other global_now literals in the clause in which the global_now literal occurs into global_next literals.

This is implemented by means of *attribute transformers*—objects that can change the attribute of a literal. Given a pair of complementary literals, we first check if these literals are “compatible” (i.e. unifiable in terms of first-order logic) and, if this is the case, a pair of attribute transformers is constructed. When the resolvent is generated, we apply the corresponding attribute transformer to every literal of the premises.

Saturation by step resolution. We implement an OTTER-like saturation method where the set of all clauses is split into an *active* and a *passive* clause set, and all inferences are performed between a clause, *selected* from the passive clause set, and the active clause set (for a detailed description see e.g. [25]). Generated clauses are simplified by subsumption and forward subsumption resolution. As on a typical run of the saturation method, the given set of clauses is satisfiable (since the set of clauses are originating from the search algorithm needed for the eventuality resolution rule), we do not employ any special clause selection and clause preference technique. Instead, passive clauses are grouped according to their maximal literal.

Indexing. In order to speed-up resolution, we group active clauses according to their maximal literal. For (multi-literal) subsumption, we employ a trie-like data structure of the same kind that is used for string matching with wild-card characters [3]. For the current implementation, the subsumption algorithm does not distinguish literals with different attributes, thus providing us only with an imperfect filter whose result is re-checked afterward. Global clauses are split into the *now* and *next* parts that are inserted into the index separately.

We give some more detaile on the subsumption indexing. Every propositional clause is represented as an *ordered* string of literals; no literal occurs more than once into the string. A set of strings (clauses) is kept in a *digital search trie* [20]. This representation has the advantage that every path in the trie is ordered and labels of outgoing edges of every node are ordered as well. A trie representation of the following set of clauses

$$1.\ (a \lor c) \quad 2.\ (a \lor b \lor c) \quad 3.\ (a \lor c \lor d) \quad 4.\ (b \lor d),$$

where $a > b > c > d$, is given in Fig. 5. We use this data structure for both *forward* and *backward* subsumption (to test if a given query clause is subsumed by an indexed clause and to find all indexed clauses subsumed by a given query clause, respectively). In a subsumption test, a query string is read from left

```
function Saturate(N)
begin
  repeat
    New := res(N);
    New := simp(New,N);
    N := N ∪ New;
  until (New = ∅ or ⊥ ∈ N);
  return N;
end
```

where $res(N)$ is the set of conclusions of inference steps by the initial and step resolution rules using the clauses in N as premises.

Fig. 3. A simple saturation procedure

```
function BFS(N_0, i, G_i, L)
begin
  N_1 := Saturate(N_0 ∪ {○L ∨ ⋁_{j=1}^{n} ○L_j | ⋁_{j=1}^{n} L_j ∈ G_i});
  if (⊥ ∈ N_1) then
     return {∅}
  else
     G_{i+1} := {⋁_{i=1}^{m} K_i | □(⋁_{i=1}^{m} K_i) ∈ (N_1 \ N_0)}
     if (G_{i+1} = ∅) then
        return ∅
     elsif (G_{i+1} ≡ G_i) then
        return G_{i+1}
     else
        return BFS(N_0, i+1, G_{i+1}, L)
     endif
  endif
end
```

where the return value $\emptyset$ indicates that no set of global clauses has been found such that eventuality resolution can be applied with literal L, and return value $\{\emptyset\}$ indicates that eventuality resolution can be applied to the empty set of global clauses for literal L.

Fig. 4. A breadth-first search algorithm for the eventuality resolution rule

to right and the index trie is traversed. The difference between forward and backward subsumptions is in how we traverse the trie.

In forward subsumption, for every outgoing branch of the current node, if the label of the branch coincides with the current character of a query string, the branch is recursively visited; alternatively we move to the next character of the query string (this is slightly improved by use of ordering which is omitted here for simplicity). If the test enters a node labeled with an indexed clause, this indexed clause subsumes the query clause. For example, a forward subsumption test for the clause $(a \vee b \vee d)$ would start at the state (1, "*abd*") (by a state we mean a pair of a node and a string) then visit the states (2, "*bd*"), (4, "*d*"), then backtrack to (1, "*abd*"), go to (3, "*d*"), and, finally, to (6, ""). Node 6 is labeled with $(b \vee d)$ which subsumes the query clause.

In backward subsumption, we have to visit all branches whose labels are greater than or equal to the current character of a query string; however, we only move to the next character of the query string if the label of a branch coincides with the current character. If the query string has been read to the end, all clauses kept below the current node are subsumed. For example, a backward subsumption test for the clause $(a \vee c)$ would start at the state (1, "*ac*") move to (2, "*c*"), then to (4, "*c*") and (7, ""); the clause $(a \vee b \vee c)$ is subsumed. After that, the test backtracks to (2, "*c*") and moves to (5, ""); clauses $(a \vee c)$ and $(a \vee c \vee d)$ are also subsumed.

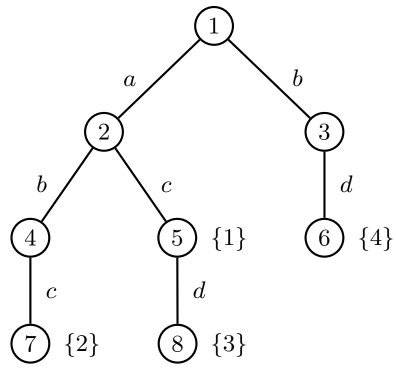

Fig. 5. Trie-based subsumption index for the clauses 1. $(a \vee c)$, 2. $(a \vee b \vee c)$, 3. $(a \vee c \vee d)$, 4. $(b \vee d)$ with the atom ordering $a > b > c > d$.

	TRP++		SPASS 2.0		Vampire 2.0		Vampire 5.0	
	median	total	median	total	median	total	median	total
uf20-91	0.02	2.14	0.02	1.90	0.04	3.95	0.01	1.42
flat30-60	11.55	2605.34	1848.95	–	10.73	–	1.80	360.90
uf50-218	197.01	27909.14	17.84	40214.27	22.68	–	1.65	211.70
uuf50-218	109.11	19153.86	49.86	12695.15	3.54	671.09	1.30	143.70
hole6		0.14		627.48		7.83		9.26
hole7		1.07		–		1216.25		1592.32
hole8		7.11		–		–		–
hole9		40.57		–		–		–
hole10		224.91		–		–		–

Fig. 6. Comparison on SAT instances

Resolution engine performance. To evaluate the performance of the step resolution inference engine of **TRP++**, we compare **TRP++** with theorem provers based on first-order resolution, SPASS 2.0[1] and two versions of Vampire[2], namely Vampire 2.0-CASC (Vampire 2.0 for short) and Vampire 5.0. Vampire 5.0 has been the winner of CASC-18 in the MIX and FOF divisions.

For the first comparison, we have taken from the SATLIB benchmark problem library[3] sets of both satisfiable (uf20-91, uf50-218, and flat30-60, each consisting of 100 problems) and unsatisfiable (uuf50-218, consisting of 100 problems) randomly generated propositional formulae in CNF form and five instances of the Pigeon-Hole principle, hole6–hole10. The tests have been performed on a PC with a 1.3GHz AMD Athlon processor, 512MB main memory, and 1GB virtual memory running RedHat Linux 7.1. For each individual satisfiability test a time-limit of 10000 CPU seconds was used. All theorem provers were used in 'auto mode', except for SPASS 2.0 where we have disabled splitting. Otherwise, SPASS 2.0 behaves like a DPPL-based SAT-solver and outperforms all other systems easily.

[1] `http://spass.mpi-sb.mpg.de/`
[2] `http://www.math.miami.edu/~tptp/CASC/`
[3] `http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html`

	TRP++		SPASS 2.0		Vampire 2.0		Vampire 5.0	
	median	total	median	total	median	total	median	total
uf20-91g	0.02	2.04	0.04	4.45	0.05	4.49	0.02	2.09
flat30-60g	11.49	2618.82	2289.01	–	17.75	–	2.95	597.10
uf50-218g	196.76	27867.16	244.86	–	34.34	–	2.30	300.60
uuf50-218g	108.57	19060.00	52.26	13552.70	5.25	936.53	1.80	202.20
ring3		0.93		–		–		4.89
ring5		7191.63		–		–		5631.53

Fig. 7. Comparison on FO instances

Figure 6 summarises the results of this first comparison. For uf20-91, flat30-60, uf50-218, uuf50-218 we give the median and total CPU time required for a problem class ('–' indicates that not all problems could be solved). Vampire 5.0 shows the overall best performance of the systems. **TRP++** is slower than the other provers on the 'easier' problems (as indicated by the median CPU time), but competes quite well with Vampire 2.0 and SPASS 2.0 on the whole (as indicated by the total CPU time). One of the reasons why the other provers beat **TRP++** on uuf50-218 is because of their clause selection and preference techniques that speed up proof search. However, it is surprising that **TRP++** performs much better than the other provers on hole8–hole10.

For the second comparison, we again used the problems in uf20-91, flat30-60, uf50-218, and uuf50-218, but this time considering the clauses as global clauses. To be able to use the first-order theorem provers on these problems, we apply the arithmetic translation to them. These test should provide some indication whether the particular data representation we have used for SNF clauses has an advantage over a 'first-order' representation. Two additional tests were conducted on the arithmetic translation of two problems, ring3 and ring5, describing the behaviour of an algorithm [13] that orients rings with 3 and 5 nodes, respectively. In order to describe the non-deterministic behaviour of the original algorithm in PLTL, eventuality clauses would be needed to which the arithmetic translation described above cannot be applied. Therefore, we used a reformulation of the two problems which avoids these clauses. Both problems are quite large (ring3 contains 24 variables and 268 clauses, ring5 contains 40 variables and 449 clauses) and unsatisfiable. Figure 7 summarises the results of this second comparison. As we can see, the performance of **TRP++** is not affected by the change from propositional to global clauses while for all other theorem provers we see a negative impact. On ring3 and ring5, **TRP++** can compete with Vampire 5.0 while the other provers fail.

Overall the results of these experiments indicate that there is still room for improvements of our implementation.

5 Comparison with other temporal provers

For comparison with other PLTL decision procedures, we selected the following systems: **TRP++**, a tableau-based procedure developed by McGuire et al. [19] which is incorporated in *STeP*, two tableau-based procedures included in the Logic Workbench 1.1, one developed by Janssen [18], the other by Schwendimann [26], and **TRP** 1.0, our previous prototype implementation of temporal resolution in SICStus Prolog 3.9.1.

We have compared the systems on two classes of randomly generated PLTL-formulae introduced in [17]. These classes, called $\mathcal{C}^1_{ran}$ and $\mathcal{C}^2_{ran}$, are intended to show the relative strength and weaknesses of tableau-based and resolution-based decision procedures for PLTL. In general, formulae in these classes are conjunctions of SNF clauses which are characterised by four parameters, n, k, p, and l where n determines the number of propositional variables in the random part of a formula, k the number of disjuncts in a random SNF clause, p the probability with which an atom occurs positively in a random SNF clause, and l the number of conjuncts in the random part of a formula.

The first class, $\mathcal{C}^1_{ran}$, is intended to show that decision procedures based on temporal resolution can show a better performance than those based on tableau calculi. To this end we use formulae with a large number of global clauses, each containing k disjuncts, and a chain of eventuality clauses such that ignoring the eventuality clauses a large number of models exists, but eventuality checks will fail in a high percentage of them. More precisely, formulae in $\mathcal{C}^1_{ran}$ have the form

$$\begin{array}{l} \quad\Box(\bigcirc L^1_1 \vee \ldots \vee \bigcirc L^1_k) \wedge \ldots \wedge \Box(\bigcirc L^l_1 \vee \ldots \vee \bigcirc L^l_k) \\ \wedge\ \Box(\neg p_1 \vee \Diamond p_2) \\ \wedge\ \Box(\neg p_2 \vee \Diamond p_3) \\ \vdots \\ \wedge\ \Box(\neg p_n \vee \Diamond p_1), \end{array}$$

where for each global clause, the literals $L^i_1, \ldots, L^i_k$ are generated by choosing k distinct variables randomly from the set $\{p_1, \ldots, p_n\}$ of n propositional variables and by determining the polarity of each literal with probability p. The eventuality clauses included in φ only depend on the parameter n.

The second class, $\mathcal{C}^2_{ran}$, is intended to show that there are also classes of formulae where tableaux-based decision procedures can perform better than those based on temporal resolution. To this end, we construct the formulae in $\mathcal{C}^2_{ran}$ in such a way that decision procedures based on the temporal resolution calculus have to make heavy use of the temporal resolution inference rule. This means we again need a set of eventuality clauses which we choose in such that way that under certain circumstances, tableaux-based decision procedures do not need to perform an eventuality check at all. More precisely, formulae in $\mathcal{C}^2_{ran}$ have the

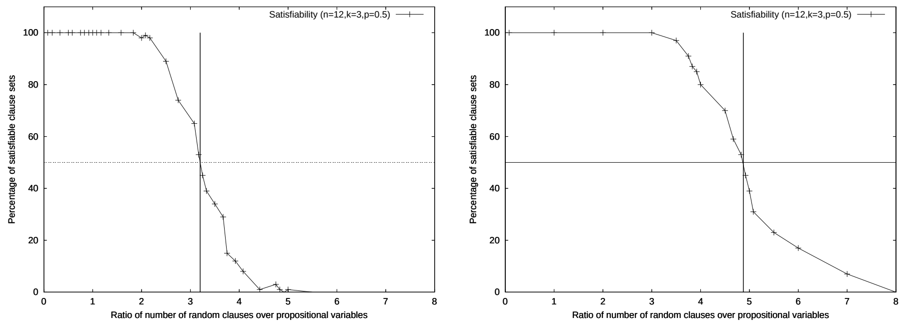

Fig. 8. Percentage of satisfiable formulae in $\mathcal{C}^1_{ran}$ and $\mathcal{C}^2_{ran}$

form

$$
\begin{array}{l}
\quad (r_1 \vee L^1_1 \vee \ldots \vee L^1_k) \wedge \ldots \wedge (r_1 \vee L^l_1 \vee \ldots \vee L^l_k) \\
\wedge \Box(\neg r_n \vee \bigcirc r_1) \\
\wedge \Box(\neg r_{n-1} \vee \bigcirc r_n) \\
\vdots \\
\wedge \Box(\neg r_1 \vee \bigcirc r_2) \\
\wedge \Box(\neg r_n \vee \bigcirc \neg q_n) \wedge \ldots \wedge \Box(\neg r_1 \vee \bigcirc \neg q_n) \\
\wedge (\neg r_1 \vee q_1) \wedge (\neg r_1 \vee \neg q_n) \\
\wedge \Box(\neg q_1 \vee \Diamond s_2) \wedge \Box(\neg s_2 \vee q_2 \vee \bigcirc q_n \vee \ldots \vee \bigcirc q_3) \\
\vdots \\
\wedge \Box(\neg q_{n-1} \vee \Diamond s_n) \wedge \Box(\neg s_n \vee q_n)
\end{array}
$$

where for each of the first l initial clauses, the literals $L^i_1, \ldots, L^i_k$ are generated by choosing k distinct variables randomly from the set $\{p_1, \ldots, p_n\}$ of n propositional variables and by determining the polarity of each literal with probability p. The global and sometime clauses included in φ only depend on the parameter n. Since formulae in both $\mathcal{C}^1_{ran}$ and $\mathcal{C}^2_{ran}$ are in conjunctive normal form and each conjunct is a SNF clause, we can consider a formulae in either class as a set of SNF clauses.

Here we will focus on just one choice of the parameters n, k, and p, namely $n = 12$, $k = 3$, and $p = 0.5$. Furthermore, we only consider formulae of $\mathcal{C}^1_{ran}$ and $\mathcal{C}^2_{ran}$ where the ratio l/n ranges from 0 to 8. For each ratio l/n that we have considered, 100 sets of SNF clauses have been generated and tested. The two graphs in Figure 8 show the percentages of satisfiable formulae in $\mathcal{C}^1_{ran}$ and $\mathcal{C}^2_{ran}$ for ratios l/n in the range from 0 to 8. We see that for a ratio $l/n = 0$ all formulae in both classes are satisfiable, the percentage of satisfiable formulae sinks monotonically with increasing ratio, and for a ratio equal to $l/n = 8$ all formulae in both classes are unsatisfiable. This last observation is the motivation for restricting ourselves to ratios $l/n \leq 8$. In the case of $\mathcal{C}^1_{ran}$, for a ratio $l/n = 3.2$ exactly half the formulae are satisfiable, while the same is true for $\mathcal{C}^2_{ran}$ for a ratio $l/n = 4.875$.

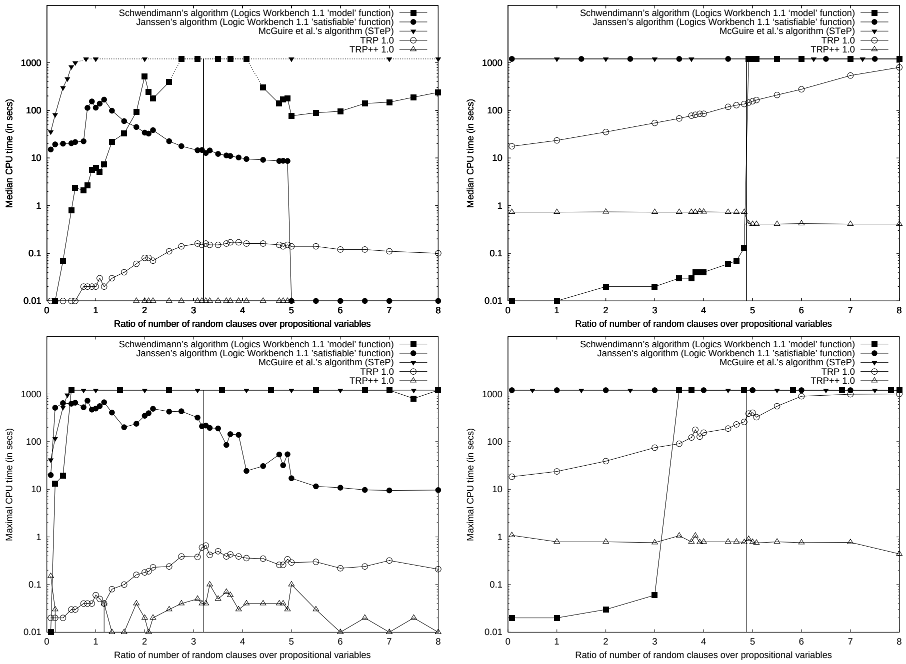

Fig. 9. Performance of the systems on $\mathcal{C}^1_{ran}$ and $\mathcal{C}^2_{ran}$

Based on the considerations in [17] we expect that resolution-based decision procedures like **TRP** and **TRP++** outperform tableau-based procedures on $\mathcal{C}^1_{ran}$ for ratios l/n between 3.2 and 5, while for $\mathcal{C}^2_{ran}$ we expect that **TRP** and **TRP++** are outperformed for ratios l/n between 0 and 4.875.

Again, the tests have been performed on a PC with a 1.3GHz AMD Athlon processor, 512MB main memory, and 1GB virtual memory running RedHat Linux 7.1. For each individual satisfiability test of a set of SNF clauses a time-limit of 1000 CPU seconds was used. The left-hand side of Figure 9 depicts the behaviour of the systems on $\mathcal{C}^1_{ran}$. A vertical line divides the graphs at the point where the number of satisfiable sets of SNF clauses equals the number of unsatisfiable ones. The right-hand side of the figure gives the same information for $\mathcal{C}^2_{ran}$. For each ratio l/n we have measured the CPU time each system has required to solve each of the 100 SNF clause sets for that ratio and computed the median. The upper part of the figure shows the resulting graphs for the median CPU time consumption of each of the systems, while the lower part of the figure shows the graphs for the maximal CPU time consumption. Note that in all performance graphs, a point for a system above the 1000 CPU second mark indicates that the median or maximal CPU time required by the system has exceeded our time-limit.

Important points to note are that **TRP** and **TRP++** perform as expected compared to the other systems and that **TRP++** performs considerably better than **TRP** on both classes indicating that the improved data representation and

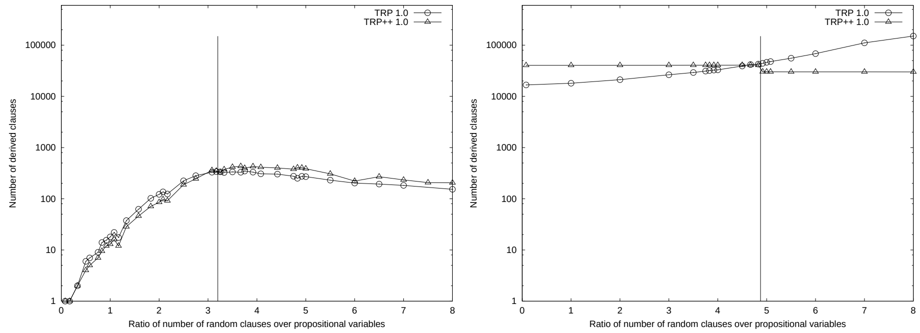

Fig. 10. Number of derived clauses for **TRP** and **TRP++** on $\mathcal{C}^1_{ran}$ and $\mathcal{C}^2_{ran}$

indexing techniques used in **TRP++** compared to **TRP** have paid off. An interesting observation is that on $\mathcal{C}^2_{ran}$ the behaviour of **TRP** and **TRP++** differs in a way that is not easily explained by these improvements alone. While the median CPU time consumption of **TRP** on $\mathcal{C}^2_{ran}$ grows steadily as the number of clauses in the clause sets under consideration increases, the median CPU time consumption of **TRP++** remains constant and shows even a significant drop at the point where the majority of clauses turns unsatisfiable. Figure 10 depicts graphs showing the median number of clauses derived by **TRP** and **TRP++** on $\mathcal{C}^1_{ran}$ (left-hand side) and $\mathcal{C}^2_{ran}$ (right-hand side). We see a good correlation to the median CPU time consumption of the two systems. We can also see that on $\mathcal{C}^1_{ran}$ both system derive roughly the same number of clauses. Thus, the difference in performance of both systems on $\mathcal{C}^1_{ran}$ is mainly due to the implementational improvements discussed before. However, on $\mathcal{C}^2_{ran}$ we see that the differing behaviour of **TRP** versus **TRP++** is reflected in differing numbers of derived clauses. It turns out that this is due to different orderings used by two systems. **TRP** uses an ordering based on the lexicographical ordering on the names of propositional variables while **TRP++**, by default, uses an ordering based on the order in which the propositional variables occur in the clause set. On unsatisfiable formulae in $\mathcal{C}^2_{ran}$, which dominate the behaviour for ratios greater than 4.875, a smaller number of applications of the eventuality rule occurs than on satisfiable formulae. Applications of this rule again lead to step resolution inferences, the number of which will depend on the ordering used. For **TRP++**, this reduction in the number of applications of the eventuality rule leads to an observable reduction in the number of derived clauses, while for **TRP** the effect is not sufficiently significant.

6 Conclusion and future work

As is evident from the empirical data presented in Sections 4 and 5, the performance of **TRP++** is considerably better than that of our prototypical system **TRP** in all our experiments. This is mainly due to the choice of programming language and data structures, in particular, the "near propositional" representation of clauses and the trie-like data structure for storing clause sets together with the algorithms for forward and backward subsumption which are based on these data structures.

While these improvements lead to a better runtime performance they do not necessarily influence the more abstract performance measure given by the number of derived clauses. However, as is evident from the graph in Figure 10 comparing the number of derived clauses for **TRP++** and **TRP** on $\mathcal{C}^2_{ran}$, also on this measure **TRP++** can outperform **TRP**. While this example shows that, as for first-order logic, orderings play an important role in improving the performance of a theorem prover, at the moment we have no heuristics which could help us to choose the most appropriate ordering for a problem.

Moreover, it can also be expected that the use of a *selection function* [2] can further reduce the number of derived clauses and may in some cases even eliminate the fundamental disadvantage that PLTL decision procedures based on temporal resolution have over tablaux-based decision procedures on classes of PLTL formulae like $\mathcal{C}^2_{ran}$.

Finally, we are currently working on extending the calculus presented in Section 3 as well as its implementation from propositional linear-time temporal logic to decidable fragments of first-order linear-time temporal logic. A first step in this direction has been described in [5].

References

1. A. Artale and E. Franconi. *Temporal description logics.* In D. Gabbay, M. Fisher, and L. Vila, editors, *Handbook of Time and Temporal Reasoning in Artificial Intelligence*. World Scientific Publishing, to appear.
2. L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In A. Robinson and A. Voronkov, *Handbook of Automated Reasoning*, pp. 19–99. Elsevier, 2001.
3. J. Bentley and R. Sedgewick. Fast algorithms for sorting and searching strings. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1997.
4. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking.* MIT Press, 1999.
5. A. Degtyarev and M. Fisher. Towards first-order temporal resolution. In *Proc. KI 2001*, volume 2174 of *LNAI*, pages 18–32. Springer, 2001.
6. C. Dixon. Search strategies for resolution in temporal logics. In *Proc. CADE-13*, volume 1104 of *LNAI*, pages 673–687. Springer, 1996.
7. C. Dixon. Using Otter for temporal resolution. In H. Barringer, M. Fisher, D. Gabbay, and G. Gough, editors, *Advances in Temporal Logic*, pages 149–166. Kluwer, 2000.
8. R. Fagin, Halpern J., Moses Y., and Vardi M. *Reasoning About Knowledge.* MIT Press, 1996.
9. M. Fisher. A resolution method for temporal logic. In J. Myopoulos and R. Reiter, editors, *Proc. IJCAI'91*, pages 99–104. Morgan Kaufman, 1991.
10. M. Fisher. An introducion to executable temporal logics. *Knowledge Engineering Review*, 11(1):43–56, 1996.
11. M. Fisher. A temporal semantics for concurrent METATEM. *J. Symbolic Computation*, 22(5/6):627–648, 1997.
12. M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, 2001.

13. J.-H. Hoepman. Uniform deterministic self-stabilizing ring-orientation on odd-length rings. In *WDAG '94, Proceedings*, volume 857 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 1994.
14. G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.
15. G. J. Holzmann. The model checker Spin. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.
16. U. Hustadt and R. A. Schmidt. Formulae which highlight differences between temporal logic and dynamic logic provers. In *Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, Technical Report DII 14/01, pages 68–76. Dipartimento di Ingegneria dell'Informazione, Unversitá degli Studi di Siena, 2001.
17. U. Hustadt and R. A. Schmidt. Scientific benchmarking with temporal logic decision procedures. In *Proc. KR2002*, pages 533–544. Morgan Kaufmann, 2002.
18. G. Janssen. *Logics for Digital Circuit Verification: Theory, Algorithms, and Applications*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1999.
19. Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In *Proc. CAV'93*, volume 697 of *LNCS*, pages 97–109. Springer, 1993.
20. D. E. Knuth. *The Art of Computer Programming. Volume III: Sorting and Searching*. Addison-Wesley, 1973.
21. O. Kupferman and M. Y. Vardi. *Synthesis with incomplete information*, pages 109–128. Kluwer, 2000.
22. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.
23. A Pnueli and R Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. Princip. Prog. Lang.*, pages 179–190, 1989.
24. A. S. Rao and M. P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293–343, June 1998.
25. A. Riazanov and A. Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, page 14, To appear.
26. S. Schwendimann. *Aspects of Computational Logic*. PhD thesis, Universität Bern, Switzerland, 1998.
27. M. P. Shanahan. *Solving the Frame Problem*. MIT Press, 1997.
28. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.
29. A. Tansel, editor. *Temporal Databases: theory, design, and implementation*. Benjamin/Cummings, 1993.
30. F. Wolter and M. Zakharyaschev. Temporalizing description logics. In *Frontiers of Combining Systems II*, pages 379–401, 2000.